

PATENT ABSTRACTS OF JAPAN

(11)Publication number : 08-292932

(43)Date of publication of application : 05.11.1996

(51)Int.Cl. G06F 15/16
G06F 9/46
G06F 9/46

(21)Application number : 08-036964

(71)Applicant : MATSUSHITA ELECTRIC IND CO LTD

(22)Date of filing : 23.02.1996

(72)Inventor : TANAKA TETSUYA
FUKUDA AKIRA
TANUMA HITOSHI

(30)Priority

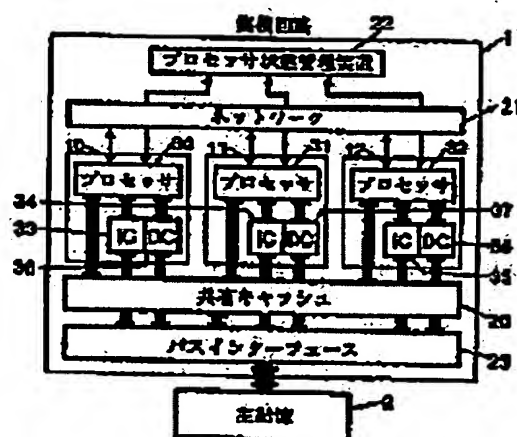
Priority number : 07 36836 Priority date : 24.02.1995 Priority country : JP

(54) MULTIPROCESSOR SYSTEM AND METHOD FOR EXECUTING TASK IN THE SAME

(57)Abstract:

PURPOSE: To provide a task executing method suitable to a task of fine granularity.

CONSTITUTION: This is a method for executing the task by the multi-processor system 1 including processors 30-32 and includes a step wherein whether or not there is a processor having a 'free state' among the processors 30-32 when a processor which is executing a task T1 among the processors 30-32 generates a new task T2, a step wherein when the processor having the 'free state' is detected, the task T2 begins to be executed by the processor by being assigned to the processor and the state of the processor is changed from the 'free state' to an 'execution state' to store a flag having a 1st value indicating that the execution of the task T1 is not interrupted; and a step wherein the execution of the task T1 is interrupted and the task T2 which is interrupted begins to be executed by the processor to store a flag having a 2nd value indicating that the execution of the task T1 has been interrupted.



LEGAL STATUS

[Date of request for examination] 26.02.1998

[Date of sending the examiner's decision of rejection] 23.08.2000

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号

特開平8-292932

(43) 公開日 平成8年(1996)11月5日

(51) Int.Cl. ⁶	識別記号	庁内整理番号	F I	技術表示箇所
G 0 6 F 15/16	4 3 0		G 0 6 F 15/16	4 3 0 B
9/46	3 4 0		9/46	3 4 0 B
	3 6 0			3 6 0 B

審査請求 未請求 請求項の数13 O L (全 16 頁)

(21) 出願番号 特願平8-36964

(22) 出願日 平成8年(1996)2月23日

(31) 優先権主張番号 特願平7-36836

(32) 優先日 平7(1995)2月24日

(33) 優先権主張国 日本 (J P)

(71) 出願人 000005821

松下電器産業株式会社

大阪府門真市大字門真1006番地

(72) 発明者 田中 哲也

大阪府門真市大字門真1006番地 松下電器
産業株式会社内

(72) 発明者 福田 晃

福岡県三潞郡城島町大字青木島199番地2

(72) 発明者 田沼 仁

大阪府三島郡島本町広瀬4丁目11番6号

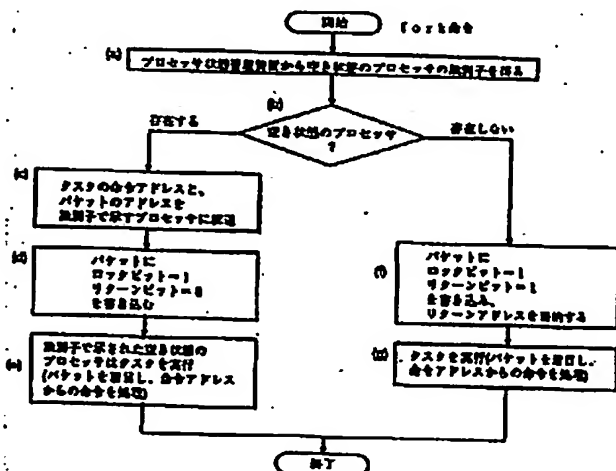
(74) 代理人 弁理士 山本 秀策

(54) 【発明の名称】 マルチプロセッサシステムおよびマルチプロセッサシステムにおいてタスクを実行する方法

(57) 【要約】

【課題】 細粒度のタスクに適したタスクの実行方法を提供する。

【解決手段】 プロセッサ30～32を含むマルチプロセッサシステム1においてタスクを実行する方法であって、プロセッサ30～32のうちタスクT1を実行中のプロセッサが新たなタスクT2を生成した場合において、プロセッサ30～32のうち「空き状態」を有するプロセッサがあるか否かを検出するステップと、「空き状態」を有するプロセッサが検出された場合には、タスクT2をそのプロセッサに割り当てることにより、そのプロセッサによるタスクT2の実行を開始し、そのプロセッサの状態を「空き状態」から「実行状態」に変更し、タスクT1の実行が中断されていないことを示す第1の値を有するフラグを格納するステップと、「空き状態」を有するプロセッサが検出されない場合には、タスクT1の実行を中断し、中断したプロセッサによるタスクT2の実行を開始し、タスクT1の実行が中断されたことを示す第2の値を有するフラグを格納するステップとを包含する方法。



1

【 特許請求の範囲】

【 請求項1 】 「 空き状態」と「 実行状態」とを有する複数のプロセッサを含むマルチプロセッサシステムにおいてタスクを実行する方法であって、

該複数のプロセッサのうち第1 タスクを実行中の第1 プロセッサが新たな第2 タスクを生成した場合において、該複数のプロセッサのうち「 空き状態」を有する第2 プロセッサがあるか否かを検出するステップと、

「 空き状態」を有する第2 プロセッサが検出された場合には、該第2 タスクを該第2 プロセッサに割り 当てることにより、該第2 プロセッサによる該第2 タスクの実行を開始し、該第2 プロセッサの状態を「 空き状態」から「 実行状態」に変更し、該第1 タスクの実行が中断されていないことを示す第1 の値を有するフラグを格納するステップと、

「 空き状態」を有する第2 プロセッサが検出されない場合には、該第1 プロセッサによる該第1 タスクの実行を中断し、該第1 プロセッサによる該第2 タスクの実行を開始し、該第1 タスクの実行が中断されたことを示す第2 の値を有するフラグを格納するステップとを包含する方法。

【 請求項2 】 前記方法は、前記第2 タスクの実行が終了した後、前記フラグが前記第1 の値と前記第2 の値のうちのいずれを有するかを判定するステップと、

前記フラグが前記第1 の値を有すると判定された場合には、前記第2 プロセッサの状態を「 実行状態」から「 空き状態」に変更するステップと、

前記フラグが前記第2 の値を有すると判定された場合には、前記第1 タスクの実行が中断されたところから前記第1 プロセッサによる前記第1 タスクの実行を再開するステップとをさらに包含する、請求項1 に記載の方法。

【 請求項3 】 前記複数のプロセッサのそれぞれは、前記複数のプロセッサを互いに識別する識別子を有しており、前記「 空き状態」を有する第2 プロセッサの検出は、該識別子を用いて行われる、請求項1 に記載の方法。

【 請求項4 】 前記複数のプロセッサのそれぞれは、タスクを割り 当てる優先順位を決定する優先度を有しており、前記第2 プロセッサへの前記第2 タスクの割り 当ては、該優先度に基づいて行われる、請求項1 に記載の方法。

【 請求項5 】 「 空き状態」と「 実行状態」とを有する複数のプロセッサを含むマルチプロセッサシステムにおいて、「 停止状態」と「 第1 実行状態」と「 第2 実行状態」とを有するタスクを実行する方法であって、該複数のプロセッサのうち第1 タスクを実行中の第1 プロセッサが新たな第2 タスクを生成した場合において、該複数のプロセッサのうち「 空き状態」を有する第2 プロセッサがあるか否かを検出するステップと、

2

「 空き状態」を有する第2 プロセッサが検出された場合には、該第2 タスクを該第2 プロセッサに割り 当てることにより、該第2 プロセッサによる該第2 タスクの実行を開始し、該第2 プロセッサの状態を「 空き状態」から「 実行状態」に変更し、該第2 タスクの状態を「 停止状態」から「 第1 実行状態」に変更するステップと、

「 空き状態」を有する第2 プロセッサが検出されない場合には、該第1 プロセッサによる該第1 タスクの実行を中断し、該第1 プロセッサによる該第2 タスクの実行を開始し、該第2 タスクの状態を「 停止状態」から「 第2 実行状態」に変更するステップとを包含する方法。

【 請求項6 】 前記方法は、前記第2 タスクの実行が終了した後、前記第2 タスクの状態を判定するステップと、前記第2 タスクが「 第1 実行状態」を有すると判定された場合には、前記第2 プロセッサの状態を「 実行状態」から「 空き状態」に変更し、前記第2 タスクの状態を「 第1 実行状態」から「 停止状態」に変更するステップと、

前記第2 タスクが「 第2 実行状態」を有すると判定された場合には、前記第2 タスクの状態を「 第2 実行状態」から「 停止状態」に変更するステップとをさらに包含する、請求項5 に記載の方法。

【 請求項7 】 前記複数のプロセッサのそれぞれは、前記複数のプロセッサを互いに識別する識別子を有しており、前記「 空き状態」を有する第2 プロセッサの検出は、該識別子を用いて行われる、請求項5 に記載の方法。

【 請求項8 】 前記複数のプロセッサのそれぞれは、タスクを割り 当てる優先順位を決定する優先度を有しており、前記第2 プロセッサへの前記第2 タスクの割り 当ては、該優先度に基づいて行われる、請求項5 に記載の方法。

【 請求項9 】 複数のタスクを並列に実行する複数のプロセッサと、

該複数のプロセッサの状態を管理し、該複数のプロセッサのそれぞれからの問い合わせに応じて「 空き状態」のプロセッサの識別子を返す状態管理手段とを備えたマルチプロセッサシステムであって、

該複数のプロセッサのそれぞれは、新たなタスクが発生した時点で、該状態管理手段に対して「 空き状態」のプロセッサがあるか否かを問い合わせる、マルチプロセッサシステム。

【 請求項1 0 】 前記状態管理手段は、該プロセッサからの問い合わせに回答して、現在の状態を次の状態に遷移させる手段と、該次の状態に基づいて該問い合わせに対する応答を出力する手段とを備えている、請求項9 に記載のマルチプロセッサシステム。

【 請求項1 1 】 前記マルチプロセッサシステムは、該複数のプロセッサのそれぞれについて、命令キャッシュ

10

20

30

40

50

メモリとデータキャッシュメモリとをさらに備えている、請求項9に記載のマルチプロセッサシステム。

【請求項12】 前記マルチプロセッサシステムは、前記複数のプロセッサ間で命令アドレスおよびパケットアドレスを転送するためのネットワークをさらに備えている、請求項9に記載のマルチプロセッサシステム。

【請求項13】 該複数のタスクのそれぞれは、細粒度である、請求項9に記載のマルチプロセッサシステム。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】本発明は、複数のタスクを並列に実行する複数のプロセッサを含むマルチプロセッサシステムおよびそのマルチプロセッサシステムにおいてタスクを実行する方法に関する。

【0002】

【従来の技術】近年、マルチプロセッサシステムは汎用計算機の並列処理による高性能化のアプローチの一つとして注目されている。マルチプロセッサシステムにおいては複数のプロセッサを一つのバスに接続し、主記憶装置を共有する共有メモリ型のマルチプロセッサシステムが主に採用されている。

【0003】このようなマルチプロセッサシステムは通常、複数のプロセッサチップをプリント基板上に実装するため、各プロセッサの処理速度に対し、プロセッサ間のバスを用いる通信や同期の処理速度は遅い。そのため、処理単位であるタスクの処理時間がプロセッサ間の通信や同期の時間に対し十分大きい場合に用いられる。この場合のタスクの大きさは中粒度～粗粒度と呼ばれ実行命令数で数1000命令程度以上とされている。このように、処理単位を大きくする（粒度を粗くする）ことでタスクの実行時間に対して相対的にプロセッサ通信や同期の時間を小さくしている。

【0004】さらに、近年半導体の集積化技術は急速に発展している。そのため、チップ内に多くの機能ユニットやメモリを搭載することができるようになってきている。マルチプロセッサシステムにおいても今後複数のプロセッサをワンチップに搭載することが可能になると思われる。その場合、プロセッサが接続されるバスもチップ内に入ることになりプロセッサ間の通信や同期の高速化はそこで実行するタスクの粒度の選択肢を広げる。即ち、タスクの大きさが細粒度、命令数で数10～数100命令程度の並列処理が可能になりつつある。今後、このような細粒度のタスクを並列処理することが主流になると予想される。近年注目されているオブジェクト指向プログラミングや関数型言語を用いたプログラミングは、いずれも「細粒度のタスクを並列処理する」ことに合致したものであるからである。

【0005】一方、マルチプロセッサシステムでは、複数のタスクを物理的に限られたプロセッサ数に割り当てることになるため、タスクの実行順序を決定し、どのプ

ロセッサに対しどのタスクを割り当てるかを適切に選択することが行われる。この処理を動的に行うため、まず実行待ちタスクを一次記憶などのタスク管理装置に格納しておき、次に空きプロセッサを検出し、空きプロセッサがある場合は、実行待ちタスクの中から実行すべきタスクを選択し、選択したタスクを空きプロセッサに割り当てること行われる。このときのタスク選択は仕事全体の実行時間を最小にするなどの目的で行われる。こういったタスクの実行順序を決定し、タスクをどのプロセッサに割り当てるかを決定する処理をスケジューリングといい、決定方法の異なるさまざまなアルゴリズムがある。また、タスク生成によって実行すべきタスクが生じた場合、タスク管理装置に実行待ちタスクとして登録する処理もある。

【0006】図12にマルチプロセッサシステムにおける、従来のプロセッサ割当方法の動作説明図を示す。図12において、プロセッサ2はタスクを生成し、実行待ちのタスクとしてタスク管理装置にタスク4を登録している。プロセッサ0はプロセッサ1が「空き状態」であることを検出すると、タスク管理装置の実行待ちのタスクをスケジューリングアルゴリズムにしたがって一つを選択し、選択されたタスクはプロセッサ0によりプロセッサ1に割り当てられる。このとき、プロセッサ0はスケジューリングの処理を、プロセッサ2はタスク登録の処理をそれぞれ行っている。

【0007】これは、例えば特開昭63-208948号公報に示すように空きプロセッサ（図12ではプロセッサ1）がタスクレディーキュー（図12ではタスク管理装置）の監視を行い、実行待ちのタスクを自動的に取り出し処理する場合でも、「空き状態」のプロセッサがスケジューリングの処理を行っている。

【0008】また、例えば特開昭62-190548号公報に示されるように、タスクを依頼した依頼プロセッサが、依頼された被依頼プロセッサでのタスクの状態を監視しておき、被依頼プロセッサがタスクの終了を検出した場合、空きプロセッサとなった被依頼プロセッサにほかのタスクを適切に選択し割り当てる方法がある。この方法においては、依頼プロセッサが被依頼プロセッサの状態を監視する処理を行っている。

【0009】前記したスケジューリング処理やタスクの登録処理、もしくは被依頼プロセッサを監視する処理はそれぞれ内容は異なるもののタスクをプロセッサに割り当て実行するまでのオーバーヘッド即ちタスク処理に付随するオーバーヘッドと考えることができる。図13はタスクの処理時間と前記したオーバーヘッドの処理時間のタイムチャートを示している。図13に示すようにタスクの粒度が中～粗粒度の場合はタスクの処理時間に対してオーバーヘッドの処理時間が相対的に小さいため、オーバーヘッドの処理時間を無視できるレベルにある。

【0010】

10

20

30

40

50

5

【 発明が解決しようとする課題】 しかしながら、上記のようなタスク処理に付随するオーバーヘッドを持つマルチプロセッサシステムにおいて、プロセッサ間の通信や同期を高速化することで細粒度の並列処理を行う場合は、タスクの処理時間に対して相対的にオーバーヘッドの処理時間が大きくなる。

【 0 0 1 1 】 図1 4 は細粒度の場合のタスクの処理時間とオーバーヘッドの処理時間のタイムチャートを示している。図1 4 に示すようにオーバーヘッドの処理時間はタスクの処理時間に比べて相対的に大きくなり、オーバーヘッドの処理時間が無視できず仕事全体としての処理時間が大きくなるという問題を有する。

【 0 0 1 2 】 本発明は上記問題点に鑑み、細粒度の並列処理をプロセッサ間の通信や同期が高速なマルチプロセッサにおいて、タスク管理やスケジューリング、タスク状態の監視を行わないことで、前記したオーバーヘッドをなくし、その代わりのプロセッサに対する動的なタスク割当を一元的、単純かつ高速に行う方法を提供することにある。

【 0 0 1 3 】

【 課題を解決するための手段】 本発明の方法は、「空き状態」と「実行状態」とを有する複数のプロセッサを含むマルチプロセッサシステムにおいてタスクを実行する方法であって、該複数のプロセッサのうち第1 タスクを実行中の第1 プロセッサが新たな第2 タスクを生成した場合において、該複数のプロセッサのうち「空き状態」を有する第2 プロセッサがあるか否かを検出するステップと、「空き状態」を有する第2 プロセッサが検出された場合には、該第2 タスクを該第2 プロセッサに割り当てることにより、該第2 プロセッサによる該第2 タスクの実行を開始し、該第2 プロセッサの状態を「空き状態」から「実行状態」に変更し、該第1 タスクの実行が中断されていないことを示す第1 の値を有するフラグを格納するステップと、「空き状態」を有する第2 プロセッサが検出されない場合には、該第1 プロセッサによる該第1 タスクの実行を中断し、該第1 プロセッサによる該第2 タスクの実行を開始し、該第1 タスクの実行が中断されたことを示す第2 の値を有するフラグを格納するステップとを包含しており、これにより上記目的が達成される。

【 0 0 1 4 】 前記方法は、前記第2 タスクの実行が終了した後、前記フラグが前記第1 の値と前記第2 の値のうちのいずれを有するかを判定するステップと、前記フラグが前記第1 の値を有すると判定された場合には、前記第2 プロセッサの状態を「実行状態」から「空き状態」に変更するステップと、前記フラグが前記第2 の値を有すると判定された場合には、前記第1 タスクの実行が中断されたところから前記第1 プロセッサによる前記第1 タスクの実行を再開するステップとをさらに包含してもよい。

6

【 0 0 1 5 】 前記複数のプロセッサのそれぞれは、前記複数のプロセッサを互いに識別する識別子を有しており、前記「空き状態」を有する第2 プロセッサの検出は、該識別子を用いて行われてもよい。

【 0 0 1 6 】 前記複数のプロセッサのそれぞれは、タスクを割り当てる優先順位を決定する優先度を有しており、前記第2 プロセッサへの前記第2 タスクの割り当ては、該優先度に基づいて行われてもよい。

【 0 0 1 7 】 本発明の他の方法は、「空き状態」と「実行状態」とを有する複数のプロセッサを含むマルチプロセッサシステムにおいて、「停止状態」と「第1 実行状態」と「第2 実行状態」とを有するタスクを実行する方法であって、該複数のプロセッサのうち第1 タスクを実行中の第1 プロセッサが新たな第2 タスクを生成した場合において、該複数のプロセッサのうち「空き状態」を有する第2 プロセッサがあるか否かを検出するステップと、「空き状態」を有する第2 プロセッサが検出された場合には、該第2 タスクを該第2 プロセッサに割り当てることにより、該第2 プロセッサによる該第2 タスクの実行を開始し、該第2 プロセッサの状態を「空き状態」から「実行状態」に変更し、該第2 タスクの状態を「停止状態」から「第1 実行状態」に変更するステップと、「空き状態」を有する第2 プロセッサが検出されない場合には、該第1 プロセッサによる該第1 タスクの実行を中断し、該第1 プロセッサによる該第2 タスクの実行を開始し、該第2 タスクの状態を「停止状態」から「第2 実行状態」に変更するステップとを包含しており、これにより上記目的が達成される。

【 0 0 1 8 】 前記方法は、前記第2 タスクの実行が終了した後、前記第2 タスクの状態を判定するステップと、前記第2 タスクが「第1 実行状態」を有すると判定された場合には、前記第2 プロセッサの状態を「実行状態」から「空き状態」に変更し、前記第2 タスクの状態を「第1 実行状態」から「停止状態」に変更するステップと、前記第2 タスクが「第2 実行状態」を有すると判定された場合には、前記第2 タスクの状態を「第2 実行状態」から「停止状態」に変更するステップとをさらに包含してもよい。

【 0 0 1 9 】 前記複数のプロセッサのそれぞれは、前記複数のプロセッサを互いに識別する識別子を有しており、前記「空き状態」を有する第2 プロセッサの検出は、該識別子を用いて行われてもよい。

【 0 0 2 0 】 前記複数のプロセッサのそれぞれは、タスクを割り当てる優先順位を決定する優先度を有しており、前記第2 プロセッサへの前記第2 タスクの割り当ては、該優先度に基づいて行われてもよい。

【 0 0 2 1 】 本発明のマルチプロセッサシステムは、複数のタスクを並列に実行する複数のプロセッサと、該複数のプロセッサの状態を管理し、該複数のプロセッサのそれぞれからの問い合わせに応じて「空き状態」のプロ

セッサの識別子を返す状態管理手段とを備えており、該複数のプロセッサのそれぞれは、新たなタスクが発生した時点で、該状態管理手段に対して「空き状態」のプロセッサがあるか否かを問い合わせる。これにより上記目的が達成される。

【0022】前記状態管理手段は、該プロセッサからの問い合わせに回答して、現在の状態を次の状態に移させる手段と、該次の状態に基づいて該問い合わせに対する回答を出力する手段とを備えていてもよい。

【0023】前記マルチプロセッサシステムは、該複数のプロセッサのそれぞれについて、命令キャッシュメモリとデータキャッシュメモリとをさらに備えていてもよい。

【0024】前記マルチプロセッサシステムは、前記複数のプロセッサ間で命令アドレスおよびパケットアドレスを転送するためのネットワークをさらに備えていてもよい。

【0025】該複数のタスクのそれぞれは、細粒度であってもよい。

【0026】

【発明の実施の形態】以下、図面を参照しながら、本発明の実施の形態を説明する。

【0027】図1は、本発明のマルチプロセッサシステム1の構成を示す。マルチプロセッサシステム1は、集積回路上にインプリメントされる。マルチプロセッサシステム1は、バスを介して主記憶装置2に接続される。

【0028】マルチプロセッサシステム1は、要素プロセッサユニット10~12を含む。要素プロセッサユニット10~12のそれぞれは、同一の構成を有している。マルチプロセッサシステム1に含まれる要素プロセッサユニットの数は、3に限定されるわけではない。マルチプロセッサシステム1は、任意の個数の要素プロセッサユニットを含み得る。

【0029】要素プロセッサユニット10~12は、それぞれ、プロセッサ30~32と命令キャッシュ(IC)33~35とデータキャッシュ(DC)36~38とを有している。命令キャッシュ(IC)は、命令を格納するためのキャッシュメモリであり、読み出し専用である。データキャッシュ(DC)は、データを格納するためのキャッシュメモリであり、読み出しと書き込みができる。

【0030】共有キャッシュ20は、要素プロセッサユニット10~12によって共有されている。命令セットやデータセットは、通常、主記憶装置2に格納されている。データセットは、必要に応じてバスインタフェース23を介して共有キャッシュ20にロードされる。共有キャッシュ20は、主記憶装置2と比較して非常に高速に動作することが好ましい。データキャッシュ(DC)と共有キャッシュ20とは、アドレスに応じて使い分けられる。例えば、アドレスが0x00000000~0

x7ffffffffffの範囲内である場合には、データキャッシュ(DC)がアクセスされ、アドレスが0x80000000~0xffffffffの範囲内である場合には、共有キャッシュ20がアクセスされる。

【0031】要素プロセッサユニット10~12は、ネットワーク21を介して相互に接続される。ネットワーク21は、要素プロセッサユニット10~12の相互間で命令アドレスやパケットアドレスを転送するために使用される。ネットワーク21は、例えば、3x3のクロスバスイッチを用いて実現することができる。

【0032】プロセッサ状態管理装置22は、プロセッサ30~32の状態を管理する。プロセッサ30~32のそれぞれは、「実行状態」および「空き状態」のいずれか一方の状態を有する。

【0033】プロセッサ30~32のそれぞれには固定された優先度が予め割り当てられている。ここでは、プロセッサ30~33は、この順番に高い優先度を有していると仮定する。優先度は、複数のプロセッサがプロセッサ状態管理装置22を同時にアクセスする場合において、その複数のプロセッサのうちのどのプロセッサにプロセッサ状態管理装置22に優先的にアクセスすることを許すかを決定するために使用される。

【0034】プロセッサ30~32のそれぞれは、プロセッサ30~32を互いに識別するための識別子(ID)を有している。典型的には、識別子(ID)は、番号によって表現される。

【0035】プロセッサ30~32のそれぞれは、その内部にパケットのアドレスを保持する。パケットのアドレスは、例えば、プロセッサ30~32の内部のレジスタ(図示せず)に保持される。これにより、プロセッサ30~32は、パケットを参照することができる。パケットの詳細は、図6を参照して後述される。

【0036】マルチプロセッサシステム1は、複数のタスクを並列に実行する機能を有する。例えば、プロセッサ30がタスクT1を実行しているのと並行して、プロセッサ31はタスクT2を実行することができる。

【0037】本明細書では、「タスク」とは、命令セットとデータセットとの組であると定義する。命令セットとデータセットとは、いずれも主記憶装置2に格納される。プロセッサ30~32のそれぞれは、命令セットから命令を逐次読み出し、読み出された命令を解釈実行する。データセットは、プロセッサ30~32が命令セットから読み出された命令を解釈実行する際、必要に応じて参照される。また、後述されるパケットは、データセットの少なくとも一部である。

【0038】図2は、タスクの概念を模式的に示す。この例では、タスク1は、命令セット1とデータセット1の組によって定義され、タスク2は、命令セット1とデータセット2の組によって定義され、タスク3は、命令セット2とデータセット3の組によって定義される。命

9

令セット1～2とデータセット1～3は、それぞれ、主記憶装置2に格納されている。

【0039】図3は、プロセッサ30～32の状態を管理するプロセッサ状態管理装置22の構成例を示す。プロセッサ状態管理装置22は、入力(REQ0～REQ2、RESET0～RESET2)にตอบสนองして出力(ID0～ID2、NMP0～NMP2)を提供する組み合わせ回路を含んでいる。その組み合わせ回路は、現在の*

10

*状態(S)と入力(REQ0～REQ2、RESET0～RESET2)とに応じて次の状態(next S)を決定し、次の状態に対応する出力(ID0～ID2、NMP0～NMP2)を提供する。現在の状態(S)から次の状態(next S)への遷移は、例えば、表1に示される状態遷移表に従って決定される。

【0040】

【表1】

S	REQ	RESET	next S	ID0	NMP0	ID1	NMP1	ID2	NMP2
001	001	000	011	01	0	--	--	--	--
010	010	000	011	--	--	00	0	--	--
100	100	000	101	--	--	--	--	00	0
011	001	000	111	10	0	--	--	--	--
011	010	000	111	--	--	10	0	--	--
011	011	000	111	10	0	--	1	--	--
101	001	000	111	01	0	--	--	--	--
101	100	000	111	--	--	--	--	01	0
101	101	000	111	01	0	--	--	--	1
110	010	000	111	--	--	00	0	--	--
110	100	000	111	--	--	--	--	00	0
110	110	000	111	--	--	00	0	--	1
111	001	000	111	--	1	--	--	--	--
111	010	000	111	--	--	--	1	--	--
111	100	000	111	--	--	--	--	--	1
111	011	000	111	--	1	--	1	--	--
111	101	000	111	--	1	--	--	--	1
111	110	000	111	--	--	--	--	--	1
111	111	000	111	--	1	--	1	--	1
011	000	001	010	--	--	--	--	--	--
011	000	010	001	--	--	--	--	--	--
011	001	010	101	10	0	--	--	--	--
011	010	001	110	--	--	10	0	--	--
101	000	001	100	--	--	--	--	--	--
101	000	100	001	--	--	--	--	--	--
101	001	100	011	01	0	--	--	--	--
101	100	001	110	--	--	--	--	01	0
110	000	010	100	--	--	--	--	--	--
110	000	100	010	--	--	--	--	--	--
110	010	100	011	--	--	00	0	--	--
110	100	010	101	--	--	--	--	00	0
111	000	001	110	--	--	--	--	--	--
111	000	010	101	--	--	--	--	--	--
111	000	100	011	--	--	--	--	--	--
111	000	011	100	--	--	--	--	--	--
111	000	101	010	--	--	--	--	--	--
111	000	110	001	--	--	--	--	--	--
111	001	010	101	--	1	--	--	--	--
111	001	100	011	--	1	--	--	--	--
111	001	110	001	--	1	--	--	--	--
111	010	001	110	--	--	--	1	--	--
111	010	100	011	--	--	--	1	--	--
111	010	101	010	--	--	--	1	--	--
111	100	001	110	--	--	--	--	--	1
111	100	010	101	--	--	--	--	--	1
111	100	011	100	--	--	--	--	--	1
111	011	100	011	--	1	--	1	--	--
111	101	010	101	--	1	--	--	--	1
111	110	001	110	--	--	--	1	--	1

【0041】図3において、Sは現在の状態、Next Sは次の状態を示す。これらの状態は、プロセッサ30～32の状態を示す。例えば、S=001は、プロセッサ30の状態が「実行状態」であり、プロセッサ31とプロセッサ32の状態が「空き状態」であることを示している。Next Sについても同様である。

【0042】図3において、REQ0～REQ2は、プロセッサ30～32からプロセッサ状態管理装置22に入力されるリクエストを表す。これらのリクエストは、「空き状態」のプロセッサの識別子を得ることをプロセッサ状態管理装置22に依頼するものである。表1では、REQ0～REQ2をまとめてREQと表記してい

11

る。例えば、REQ=101は、REQ0が1（アサート）であり、REQ1が0（ネゲート）であり、REQ2が1（アサート）であることを示している。

【0043】図3において、RESET0～RESET2は、プロセッサ30～32からプロセッサ状態管理装置22に入力されるリセットを表す。これらのリセットは、プロセッサ状態管理装置22内に保持されているプロセッサ30～32の状態を「実行状態」から「空き状態」に変更することをプロセッサ状態管理装置22に依頼するものである。表1では、RESET0～RESET2をまとめてRESETと表記している。例えば、RESET=010は、RESET0が0（ネゲート）であり、RESET1が1（アサート）であり、RESET2が0（ネゲート）であることを示している。

【0044】図3において、ID0～ID2は、プロセッサ30～32からのリクエストに対して「空き状態」のプロセッサの識別子を通知する信号を表す。これらの信号は、プロセッサ30～32からのリクエストに応答してプロセッサ状態管理装置22から出力される。ID0～ID2の値の意味は、以下のとおりである。

【0045】00：プロセッサ30が「空き状態」である。

【0046】01：プロセッサ31が「空き状態」である。

【0047】10：プロセッサ32が「空き状態」である。

【0048】図3において、NMP0～NMP2は、プロセッサ30～32からのリクエストに対して「空き状態のプロセッサが存在しない」旨を通知する信号を表す。これらの信号は、プロセッサ30～32からのリクエストに応答してプロセッサ状態管理装置22から出力される。NMP0～NMP2の値の意味は、以下のとおりである。

【0049】0：「空き状態」のプロセッサが存在する。「空き状態」のプロセッサの識別子は、ID0～ID2の値によって示される。

【0050】1：「空き状態」のプロセッサが存在しない。この場合、ID0～ID2の値は、don't careである。

【0051】以下、図4と図5とを参照して、プロセッサ状態管理装置22の機能および動作を説明する。プロセッサ状態管理装置22は、マルチプロセッサシステムに含まれるすべてのプロセッサの状態を管理する。具体的には、プロセッサ状態管理装置22は、プロセッサの識別子とプロセッサの状態とを一対にしてプロセッサ状態管理装置22内に保持する。プロセッサの識別子は、複数のプロセッサを互いに識別するために使用される。典型的には、プロセッサの識別子は整数で表現される。プロセッサの状態は、「実行状態」か「空き状態」かのいずれかである。

12

【0052】プロセッサ状態管理装置22は、あるプロセッサからのリクエストに応答して、「空き状態」のプロセッサが存在するか否かを判定する。「空き状態」のプロセッサが存在した場合には、プロセッサ状態管理装置22は、その「空き状態」のプロセッサの識別子をそのリクエストを発したプロセッサに返す。「空き状態」のプロセッサが存在しなかった場合には、プロセッサ状態管理装置22は、「空き状態のプロセッサが存在しない」旨のメッセージをそのリクエストを発したプロセッサに返す。

【0053】「空き状態」のプロセッサが複数個存在する場合には、プロセッサ状態管理装置22は、「空き状態」の複数のプロセッサのうち優先度の最も高いプロセッサの識別子をそのリクエストを発したプロセッサに返す。また、複数のプロセッサからのリクエストが同時にプロセッサ状態管理装置22に到達した場合には、そのリクエストを発した複数のプロセッサのうち優先度の高いものから順に上述した処理が行われる。

【0054】図4（a）および（b）は、プロセッサ状態管理装置22の動作の一例を示す。プロセッサ状態管理装置22は、4つのプロセッサ0～3の状態を管理している。図4（a）に示す例では、プロセッサ0とプロセッサ1の状態は「実行状態」であり、プロセッサ2とプロセッサ3の状態は「空き状態」である。プロセッサ0からのリクエストとプロセッサ1からのリクエストがプロセッサ状態管理装置22に入力される。

【0055】プロセッサ状態管理装置22は、プロセッサ0からのリクエストに応答して、「空き状態」のプロセッサ2の識別子をプロセッサ0に返し、プロセッサ1からのリクエストに応答して、「空き状態」のプロセッサ3の識別子をプロセッサ1に返す（図4（b）参照）。「空き状態」のプロセッサの識別子は、プロセッサの優先度に従って返される。また、プロセッサ状態管理装置22は、プロセッサ状態管理装置22内に保持されているプロセッサ2の状態を「空き状態」から「実行状態」に変更し、プロセッサ3の状態を「空き状態」から「実行状態」に変更する。

【0056】図5（a）および（b）は、プロセッサ状態管理装置22の動作の他の一例を示す。プロセッサ状態管理装置22は、4つのプロセッサ0～3の状態を管理している。図5（a）に示す例では、プロセッサ0とプロセッサ1とプロセッサ2の状態は「実行状態」であり、プロセッサ3の状態は「空き状態」である。プロセッサ0からのリクエストとプロセッサ1からのリクエストがプロセッサ状態管理装置22に入力される。

【0057】プロセッサ状態管理装置22は、プロセッサ0からのリクエストに応答して、「空き状態」のプロセッサ3の識別子をプロセッサ0に返し、プロセッサ1からのリクエストに応答して、「空き状態のプロセッサが存在しない」旨のメッセージをプロセッサ1に返す

13

(図5 (b) 参照) 。 「 空き状態のプロセッサが存在しない」 旨のメッセージは、例えば、プロセッサ状態管理装置2 2 から出力されるリターンコードの値によって表される。「 空き状態」 のプロセッサの識別子は、プロセッサの優先度に従って返される。また、プロセッサ状態管理装置2 2 は、プロセッサ状態管理装置2 2 内に保持されているプロセッサ3 の状態を「 空き状態」 から「 実行状態」 に変更する。

【 0 0 5 8 】 図4 と 図5 に示される例では、プロセッサ状態管理装置2 2 によって管理されるプロセッサの数は 4 である。しかし、これは、説明の便宜上のためであり、本発明が4 つのプロセッサを有するマルチプロセッサシステムに限定されるわけではない。本発明は、任意の数のプロセッサを含むマルチプロセッサシステムに適用され得る。

【 0 0 5 9 】 図6 は、パケット 5 0 の構成を示す。パケット 5 0 は、ロックビットを格納するロックビット領域 5 1 と、リターンビットを格納するためのリターンビット領域 5 2 と、リターンアドレスを格納するためのリターンアドレス領域 5 3 と、引数を格納するための引数領域 5 4 と、戻り 値を格納するための戻り 値領域 5 5 とを有している。パケット 5 0 は、タスク毎に共有メモリ 2 0 上に確保され、タスクに所有される。これ以降、「 タスクに所有されたパケット」 を単に「 タスクのパケット」 と呼ぶ。パケット 5 0 は、タスク間のデータの受け渡しやタスクの情報を保持するために使用される。

【 0 0 6 0 】 パケット 5 0 のロックビット 領域 5 1 には、ロックビットが格納される。ロックビットは、パケット 5 0 を所有するタスクが実行中である間、他のタスクからその実行中のタスクへのアクセスを禁止するか否かを示す。ロックビットが " 1 " であることは、アクセスを禁止していることを示す。ロックビットが " 0 " であることは、アクセスを禁止していないことを示す。

【 0 0 6 1 】 パケット 5 0 のリターンビット 領域 5 2 には、リターンビットが格納される。リターンビットは、パケット 5 0 を所有するタスクを実行する前に、他のタスクを中断したか否かを示す。リターンビットが " 0 " であることは、「 パケット 5 0 を所有するタスクを実行する前に、他のタスクを中断していない」 ことを示す。これは、「 空き状態」 のプロセッサにパケット 5 0 を所有するタスクが割り 当てられた場合に相当する。リターンビットが " 1 " であることは、「 パケット 5 0 を所有するタスクを実行する前に、他のタスクを中断した」 ことを示す。これは、「 空き状態」 のプロセッサが存在しなかったため、タスクを実行中のプロセッサがそのタスクの実行を中断して、パケット 5 0 を所有する別のタスクを実行する場合に相当する。

【 0 0 6 2 】 パケット 5 0 のリターンアドレス領域 5 3 には、リターンアドレスが格納される。リターンアドレスは、リターンビットが " 1 " である場合にのみ参照さ

14

れる。リターンアドレスは、中断されたタスクへの戻りアドレスを示す。

【 0 0 6 3 】 パケット 5 0 の引数領域 5 4 には、パケット 5 0 を所有するタスクへの引数が格納される。

【 0 0 6 4 】 パケット 5 0 の戻り 値領域 5 5 には、パケット 5 0 を所有するタスクの実行結果である戻り 値が格納される。

【 0 0 6 5 】 図7 は、プロセッサ 3 0 ~ 3 2 が fork 命令を解釈実行する手順を示す。プロセッサ 3 0 ~ 3 2 は、主記憶装置 2 に格納されている命令セットから命令を読み出す。読み出された命令が fork 命令である場合には、プロセッサ 3 0 ~ 3 2 は、図7 に示す処理を実行する。

【 0 0 6 6 】 以下、図7 を参照して、プロセッサ 3 0 が fork 命令を解釈実行する手順をステップごとに詳細に説明する。他のプロセッサ 3 1 および 3 2 が fork 命令を解釈実行する場合も同様である。なお、 fork 命令は、オペランドとして、新たなタスクの処理内容を示す命令列の先頭アドレス (以降、単に命令アドレスという) と新たなタスクのパケット 5 0 のアドレス (以降、単にパケットアドレスという) とをとる。

【 0 0 6 7 】 ステップ (a) : プロセッサ 3 0 は、「 空き状態」 のプロセッサが存在するか否かをプロセッサ状態管理装置 2 2 に問い合わせる。このような問い合わせは、例えば、プロセッサ 3 0 がプロセッサ状態管理装置 2 2 にリクエスト (REQ 0 = 1) を送ることにより達成される。プロセッサ状態管理装置 2 2 は、そのリクエストに応答して「 空き状態」 のプロセッサが存在するか否かを判定する。

【 0 0 6 8 】 「 空き状態」 のプロセッサが存在する場合には、プロセッサ状態管理装置 2 2 は、その「 空き状態」 のプロセッサの識別子をプロセッサ 3 0 に返す。

「 空き状態」 のプロセッサの識別子は、例えば、プロセッサ 3 0 がプロセッサ状態管理装置 2 2 から出力される I D 0 の値を参照することによって得られる。「 空き状態」 のプロセッサが複数個存在する場合には、優先度の最も高いプロセッサの識別子が得られる。また、複数のプロセッサが同時に fork 命令を解釈実行する場合には、優先度の高いプロセッサから順に fork 命令を解釈実行する。このようにして、プロセッサ 3 0 は、「 空き状態」 のプロセッサの識別子を取得する。

【 0 0 6 9 】 「 空き状態」 のプロセッサが存在しない場合には、プロセッサ状態管理装置 2 2 は、「 空き状態のプロセッサが存在しない」 旨のメッセージをプロセッサ 3 0 に返す。「 空き状態のプロセッサが存在しない」 旨のメッセージは、例えば、プロセッサ 3 0 がプロセッサ状態管理装置 2 2 から出力される NMP 0 の値を参照することによって得られる。

【 0 0 7 0 】 ステップ (b) : 「 空き状態」 のプロセッサが存在した場合には、プロセッサ 3 0 は、ステップ

15

(c) ~ (e) の処理を行う。「空き状態」のプロセッサが存在しない場合には、プロセッサ30は、ステップ(f) ~ (g) の処理を行う。

【 0 0 7 1 】ステップ(c) : ここでは、「空き状態」のプロセッサは、プロセッサ31であると仮定する。この場合、プロセッサ30は、fork 命令のオペランドとして与えられたタスクの命令アドレスとタスクの packets 121 を介してプロセッサ31に転送する。

【 0 0 7 2 】ステップ(d) : プロセッサ30は、fork 命令のオペランドとして与えられたタスクの packets 121 によって指定される packets 50 のロックビット領域51に" 1 " を書き込み、リターンビット領域52に" 0 " を書き込む。その後、プロセッサ30は、fork 命令の処理を完了し、次の命令の処理を行う。

【 0 0 7 3 】ステップ(e) : プロセッサ31は、ネットワーク21を介してプロセッサ30からタスクの命令アドレスとタスクの packets 121 を受け取る。プロセッサ31は、受け取った packets 121 によって指定される packets 50 を参照しながら、受け取った命令アドレスによって指定される命令から処理を開始する。

【 0 0 7 4 】以上のステップ(a) ~ (e) により、プロセッサ30は、プロセッサ31によって実行される処理とは異なる処理を独立に実行することとなる。すなわち、プロセッサ30とプロセッサ31とによって並列処理が開始される。fork 命令の処理はここで終了する。

【 0 0 7 5 】ステップ(f) : プロセッサ30は、fork 命令のオペランドとして与えられたタスクの packets 121 によって指定される packets 50 のロックビット領域51に" 1 " を書き込み、リターンビット領域52に" 1 " を書き込む。また、fork 命令の次の命令のアドレスをリターンアドレス領域53に書き込む。プロセッサ30は、実行中のタスクを中断する。

【 0 0 7 6 】ステップ(g) : プロセッサ30は、fork 命令のオペランドとして与えられたタスクの packets 121 によって指定される packets 50 を参照しな *

16

*がら、fork 命令のオペランドとして与えられたタスクの命令アドレスによって指定される命令から処理を開始する。fork 命令の処理はここで終了する。

【 0 0 7 7 】以下、図8を参照して、プロセッサ30が unlock 命令を解釈実行する手順をステップごとに詳細に説明する。他のプロセッサ31および32が unlock 命令を解釈実行する場合も同様である。

【 0 0 7 8 】ステップ(h) : プロセッサ30は、実行中のタスクが所有する packets 50 のリターンビット領域52の値が" 0 " であるか否かを判定する。リターンビット領域52の値が" 0 " であることは、プロセッサ30が処理を中断したタスクが存在しないことを示す。従って、リターンビット領域52の値が" 0 " である場合には、プロセッサ30は、ステップ(i) の処理を行う。リターンビット領域52の値が" 1 " であることは、プロセッサ30が処理を中断したタスクが存在することを示す。従って、リターンビット領域52の値が" 1 " である場合には、プロセッサ30は、ステップ(j) の処理を行う。

【 0 0 7 9 】ステップ(i) : プロセッサ30は、実行中のタスクが所有する packets 50 のロックビット領域51に" 0 " を書き込み、プロセッサ30の状態を「空き状態」にする。「空き状態」となったプロセッサ30は、これ以降の処理を行わない。unlock 命令の処理はここで終了する。

【 0 0 8 0 】ステップ(j) : プロセッサ30は、実行中のタスクが所有する packets 50 のロックビット領域51に" 0 " を書き込む。さらに、プロセッサ30は、リターンアドレス領域53に格納されているアドレスからの命令を処理することにより、中断されたタスクを復帰させる。unlock 命令の処理はここで終了する。

【 0 0 8 1 】表2は、fork 命令および unlock 命令の解釈実行に回答して、マルチプロセッサシステムの状態がどのように遷移するかを示す。表2に示される例では、マルチプロセッサシステムは、プロセッサP1とプロセッサP2とを有していると仮定する。

【 0 0 8 2 】

【 表2 】

イベント	現在の状態				次の状態			
	P1	P2	T1	T2	P1	P2	T1	T2
P1.fork	RUN T1	IDLE	EX1	STOP	RUN T1	RUN T2	EX1	EX1
P2.unlock	RUN T1	RUN T2	EX1	EX1	RUN T1	IDLE	EX1	STOP
P1.fork	RUN T1	RUN other	EX1	STOP	RUN T2	RUN other	EX1	EX2
P1.unlock	RUN T2	RUN other	EX1	EX2	RUN T1	RUN other	EX1	STOP

【 0 0 8 3 】図9に示されるように、マルチプロセッサ システムの状態は、プロセッサの状態とタスクの状態と

に区分される。

【0084】プロセッサは、2つの状態を有する。一方の状態は「空き状態(I D L E)」であり、他方の状態は「実行状態(R U N)」である。これらの状態は、プロセッサ状態管理装置22によって管理されている状態と同じものである。プロセッサの状態が「実行状態(R U N)」である場合には、そのプロセッサはいずれかのタスクを実行中である。

【0085】タスクは、3つの状態を有する。1つ目の状態は「停止状態(S T O P)」であり、2つ目の状態は「第1実行状態(E X 1)」であり、3つ目の状態は「第2実行状態(E X 2)」である。「停止状態(S T O P)」は、プロセッサがタスクの実行を待っている状態であるかタスクの実行を終了した状態である。「第1実行状態(E X 1)」は、他のタスクの実行を中断することなく現在のタスクが実行されている状態である。「第2実行状態(E X 2)」は、他のタスクの実行を中断してその後現在のタスクが実行されている状態である。プロセッサの状態が「実行状態(R U N)」である場合には、そのプロセッサに実行されているタスクの状態は、「第1実行状態(E X 1)」と「第2実行状態(E X 2)」のうちのいずれかである。

【0086】表2を再び参照して、マルチプロセッサシステムの状態がどのように遷移するかを説明する。マルチプロセッサシステムの状態は、イベントの発生にตอบสนองして、そのイベントと現在の状態に基づいて次の状態に遷移する。ここで、「P x . f o r k」という表記は、「プロセッサP x がf o r k 命令を実行した」というイベントが発生したことを表し、「P x . u n l o c k」という表記は、「プロセッサP x がu n l o c k 命令を実行した」というイベントが発生したことを表す。

【0087】表2の第1行は、プロセッサP1が「実行状態」(タスクT1を実行中)であり、プロセッサP2が「空き状態」であり、タスクT1が「第1実行状態」であり、タスクT2が「停止状態」である場合において、「プロセッサP1がf o r k 命令を実行した」というイベントにตอบสนองして、プロセッサP2の状態が「空き状態」から「実行状態」(タスクT2を実行中)に変更され、タスクT2の状態が「停止状態」から「第1実行状態」に変更されることを示す。このように状態が遷移するのは、新たなタスクT2が生成された時点でタスクT2が「空き状態」のプロセッサP2に割り当てられるからである。

【0088】表2の第2行は、表2の第1行における次の状態が現在の状態である場合において、「プロセッサP2がu n l o c k 命令を実行した」というイベントにตอบสนองして、プロセッサP2の状態が「実行状態」(タスクT2を実行中)から「空き状態」に変更され、タスクT2の状態が「第1実行状態」から「停止状態」に変更されることを示す。

【0089】表2の第3行は、プロセッサP1が「実行状態」(タスクT1を実行中)であり、プロセッサP2が「実行状態」(他のタスクを実行中)であり、タスクT1が「第1実行状態」であり、タスクT2が「停止状態」である場合において、「プロセッサP1がf o r k 命令を実行した」というイベントにตอบสนองして、プロセッサP1の状態が「実行状態」(タスクT1を実行中)から「実行状態」(タスクT2を実行中)に変更され、タスクT2の状態が「停止状態」から「第2実行状態」に変更されることを示す。このように状態が遷移するのは、新たなタスクT2が生成された時点で「空き状態」のプロセッサが存在しないため、プロセッサP1がタスクT1の実行を中断してタスクT2の実行を開始するからである。

【0090】表2の第4行は、表2の第3行における次の状態が現在の状態である場合において、「プロセッサP1がu n l o c k 命令を実行した」というイベントにตอบสนองして、プロセッサP1の状態が「実行状態」(タスクT2を実行中)から「実行状態」(タスクT1を実行中)に変更され、タスクT2の状態が「第2実行状態」から「停止状態」に変更されることを示す。

【0091】以下、f o r k 命令とu n l o c k 命令を含むプログラムを並列処理する場合におけるマルチプロセッサシステム1の動作を説明する。

【0092】図10は、1から4までの和(1+2+3+4)を二分木に基づいて計算するプログラムの手順を示す。このプログラムは、mainとsumの2つの部分に分かれており、mainは主プログラム、sumは再帰呼び出し可能でかつ並列処理可能なサブルーチンである。sumはnとmの2つの引数を取り、n+1からmまでの和を求めるものである。mainはn=0、m=4を引数としてsumを呼び出すものである。

【0093】まず、初期状態として、プロセッサ30はmainを実行していると仮定する。プロセッサ30の状態は「実行状態」である。また、プロセッサ31およびプロセッサ32の状態は「空き状態」とであると仮定する。

【0094】以下、プログラムの各ステップ(A)～(H)について、マルチプロセッサシステム1がどのように動作するかを詳細に説明する。

【0095】ステップ(A)：プロセッサ30は、n=0、m=4を引数としてsumサブルーチンを実行する。具体的には、プロセッサ30は、共有キャッシュメモリ20上にパケット50(Pk1)を確保し、そのパケット50(Pk1)の引数領域54に値0と値4とを格納する。次に、プロセッサ30は、sumの命令の先頭アドレスとパケット50(Pk1)の先頭アドレスとをオペランドとして、exec命令を実行する。exec命令とは、図7に示すf o r k 命令の処理手順のうちステップ(f)と(g)のみに対応する命令である。e

19

`exec` 命令は、`fork` 命令と同様にして、オペランドとしてタスクの命令アドレスとタスクのポケットアドレスとをとる。

【0096】プロセッサ30は、ポケット50 (Pk1) のロックビット領域51に"1"を書き込み、ポケット50 (Pk1) のリターンビット領域52に"1"を書き込み、リターンアドレス領域53に`exec` 命令の次の命令のアドレスを格納する(図7のステップ(f)を参照)。また、プロセッサ30は、ポケット50 (Pk1) を参照しながら`sum` の命令の実行を開始する(図7のステップ(g)を参照)。

【0097】ステップ(B)：プロセッサ30は、ポケット50 (Pk1) から引数 n と引数 m とを読み出し、 $(n+1)$ と m とを比較する。 $(n+1)$ と m が等しい場合には、処理はステップ(G)に進み、その他の場合には、処理はステップ(C)に進む。`sum` サブルーチンが`main` から最初に呼ばれた場合には、 $n=0$ 、 $m=4$ であるから、 $(n+1)$ と m とは等しくない。従って、処理は、ステップ(C)に進む。

【0098】ステップ(C)：プロセッサ30は、 $k=(n+m) \div 2$ を計算する。ここで、 $(n+m)=4$ であるから、 $k=2$ となる。

【0099】ステップ(D)：プロセッサ30は、 n と k とを引数として`sum` サブルーチンを実行する。具体的には、プロセッサ30は、共有キャッシュメモリ20上にポケット50 (Pk2) を確保し、そのポケット50 (Pk2) の引数領域54に値 $n(=0)$ と値 $k(=2)$ とを格納する。次に、プロセッサ30は、`sum` の命令の先頭アドレスとポケット50 (Pk2) の先頭アドレスとをオペランドとして、`fork` 命令を実行する。

【0100】プロセッサ31とプロセッサ32はいずれも「空き状態」である。プロセッサ30は、優先度に従って「空き状態」のプロセッサ31の識別子を得る(図7のステップ(a)を参照)。プロセッサ30は、タスクの命令アドレスとタスクのポケットアドレスとをプロセッサ31に転送する(図7のステップ(b)を参照)。プロセッサ30は、ポケット50 (Pk2) のロックビット領域51に"1"を書き込み、ポケット50 (Pk2) のリターンビット領域52に"0"を書き込む(図7のステップ(d)を参照)。さらに、プロセッサ31は、ポケット50 (Pk2) を参照しながら`sum` の命令の実行を開始する(図7のステップ(e)を参照)。このようにして、プロセッサ30とプロセッサ31とは`sum` サブルーチンを並列に実行する。

【0101】ステップ(E)：プロセッサ30は、 k と m とを引数として`sum` サブルーチンを実行する。具体的には、プロセッサ30は、共有キャッシュメモリ20上にポケット50 (Pk3) を確保し、そのポケット50 (Pk3) の引数領域54に値 $k(=2)$ と値 $m(=$

20

4)とを格納する。次に、プロセッサ30は、`sum` の命令の先頭アドレスとポケット50 (Pk3) の先頭アドレスとをオペランドとして、`exec` 命令を実行する。プロセッサ30が`exec` 命令の実行を開始する前に、ポケット50 (Pk1) はスタック領域に退避される。

【0102】プロセッサ30は、ポケット50 (Pk3) のロックビット領域51に"1"を書き込み、ポケット50 (Pk3) のリターンビット領域52に"1"を書き込み、リターンアドレス領域53に`exec` 命令の次の命令のアドレスを格納する(図7のステップ(f)を参照)。また、プロセッサ30は、ポケット50 (Pk3) を参照しながら`sum` の命令の実行を開始する(図7のステップ(g)を参照)。

【0103】ステップ(F)：プロセッサ30は、ステップ(E)において呼び出した`sum` サブルーチンの実行を終了した後、スタック領域に退避したポケット50 (Pk1) を復帰させる。その後、プロセッサ30は、 $s1$ と $s2$ とを加算する。ここで、 $s1$ は、ステップ(D)において実行された`sum` サブルーチンの結果を示す。従って、 $s1$ は、ポケット50 (Pk2) の戻り値領域55に格納される。 $s2$ は、ステップ(E)において実行された`sum` サブルーチンの結果を示す。従って、 $s2$ は、ポケット50 (Pk3) の戻り値領域55に格納される。プロセッサ30がステップ(E)において呼び出した`sum` サブルーチンの実行を終了した時点では、ポケット50 (Pk2) を所有するタスクはまだ実行中である可能性がある。プロセッサ30は、ポケット50 (Pk2) を所有するタスクの実行が終了した後、ポケット50 (Pk2) の戻り値領域55に格納されている値を読み出し、その値を $s1$ とする。ここでは、 $s1=3$ である。ポケット50 (Pk2) を所有するタスクの実行が終了したか否かは、ポケット50 (Pk2) のロックビット領域51の値を参照することにより判定される。ポケット50 (Pk2) のロックビット領域51の値が"0"であることは、ポケット50 (Pk2) を所有するタスクの実行が終了したことを示す。

【0104】同様にして、プロセッサ30は、ポケット50 (Pk3) を所有するタスクの実行が終了した後、ポケット50 (Pk3) の戻り値領域55に格納されている値を読み出し、その値を $s2$ とする。ここでは、 $s2=7$ である。プロセッサ30は、 $s1+s2$ を計算する。その結果、 $s=10$ が得られる。

【0105】ステップ(H)：プロセッサ30は、 s の値をポケット50 (Pk1) の戻り値領域55に格納する。その後、プロセッサ30は、`unlock` 命令を実行する。

【0106】プロセッサ30は、ポケット50 (Pk1) のリターンビット領域52に格納されている値が"1"であるか否かを判定する(図8のステップ(h)を

参照)。は、" 1 " である。従って、プロセッサ3 0 は、パケット5 0 (P k 1) のロックビット領域5 1 に" 0 " を格納し、リターンアドレス領域5 3 に格納されているアドレスからの命令を実行する(図8 のステップ(j) を参照)。この場合、main のステップ(A) の次の命令から処理が再開される。

【0 1 0 7】ステップ(G) : ステップ(B) において、 $n + 1 = m$ であると判定された場合は、処理はステップ(G) に進む。プロセッサ3 0 は、s に引数m の値を代入する。その後、処理はステップ(H) に進む。

【0 1 0 8】ここで、ステップ(D) において呼び出されたsum サブルーチンやステップ(E) において呼び出されたsum サブルーチンにおいても、上述したステップ(B) ~ (H) が実行されることに注意されたい。sum サブルーチンは、再帰呼び出し可能なサブルーチンだからである。

【0 1 0 9】このように、sum サブルーチンを再帰的に呼び出すことにより、1 から4 の和($1 + 2 + 3 + 4$) を並列に計算することが達成される。この例では、ステップ(D) におけるfork 命令とステップ(E) におけるexec 命令によって2 つのタスクが生成されている。fork 命令は「空き状態」のプロセッサがある限りそのプロセッサにタスクを割り当てるために使用される命令であり、exec 命令は、あくまで自プロセッサにタスクを割り当てるために使用される命令である。

【0 1 1 0】図1 1 は、上述した処理の内容を模式的に示したものである。図1 1 に示されるように、タスクsum (0 , 4) からfork 命令とexec 命令により2 つのタスクsum (0 , 2) とタスクsum (2 , 4) とが生成される。タスクsum (0 , 2) はプロセッサ3 1 に割り当てられ、タスクsum (2 , 4) はプロセッサ3 0 に割り当てられる。同様に、2 つのタスクのそれぞれからさらに2 つのタスクが生成される。「空き状態」のプロセッサが存在する限り他のプロセッサにタスクが割り当てられる。

【0 1 1 1】タスクsum (2 , 4) からタスクsum (2 , 3) とタスクsum (3 , 4) とが生成される。しかし、いずれのタスクもプロセッサ3 0 に割り当てられる。タスク(2 , 3) の割り当て時に「空き状態」のプロセッサがすでに存在しなくなっているからである。

【0 1 1 2】このように、本発明のマルチプロセッサシステム1 におけるプロセッサ3 0 ~ 3 2 のそれぞれは、fork 命令を解釈実行することにより、「空き状態」のプロセッサが存在する場合にはそのプロセッサにタスクを割り当て、「空き状態」のプロセッサが存在しない場合には実行中のタスクの実行を中断して、そのプロセッサにタスクを割り当てる。このようにして、処理すべきタスクが生成されると同時に「空き状態」のプロセッサか、あるいはタスクを生成したプロセッサのいずれか

にその生成されたタスクが割り当てられる。その結果、生成されたタスクは即時に実行される。これにより、従来のマルチプロセッサシステムでは必要とされた処理すべきタスクを保存する機構や、タスクの実行順序をスケジューリングする機構は不要となる。また、「空き状態」のプロセッサが存在する場合には、必ずそのプロセッサにタスクが割り当てられるため、プロセッサの利用効率も高い。

【0 1 1 3】さらに、fork 命令やunlock 命令は簡単なハードウェアで実現することができ、高速な処理も実現することができる。

【0 1 1 4】従って、集積回路上に実装されたマルチプロセッサシステム1 において、例示した0 から4 までの和を求めるプログラムのような、タスクの処理時間がスケジューリング処理時間や実行待ちタスクの管理処理に要する時間に比べて小さいプログラムを並列処理する場合には、本発明のタスク実行方法は非常に有用である。

【0 1 1 5】なお、集積回路の外部から割り込みが入った場合には、プロセッサ状態管理装置2 2 を用いて「空き状態」のプロセッサを検出し、「空き状態」のプロセッサのうち最も優先度の低いプロセッサに割り込み処理を行わせることにより、割り込み処理による性能低下を低減できる。

【0 1 1 6】なお、集積回路のプロセッサがすべて「空き状態」になったことは、プロセッサ状態管理装置2 2 を用いて検出することができる。従って、この場合には、いずれかのプロセッサで例外処理を行うことによりデッドロックを回避することができる。

【0 1 1 7】

【発明の効果】以上のように、本発明によれば、あるプロセッサで新たなタスクを生成したときにそのタスクの実行を他あるいは自プロセッサによりただちに開始することができる。このことは、タスクを保持しておく機構やタスクの実行順序をスケジューリングする機構を不要にする。また、実行待ちのタスクを選択し、その選択されたタスクを「空き状態」のプロセッサに割り当てる処理も不要となる。

【0 1 1 8】その結果、タスクの処理時間に比較してプロセッサ割り当てに要する時間が少なくてすむ。これにより、マルチプロセッサシステムにおいて、粒度の細かい並列処理の高速化を図ることができる。

【図面の簡単な説明】

【図1】本発明のマルチプロセッサシステム1 の構成を示す図である。

【図2】タスクの概念を模式的に示す図である。

【図3】マルチプロセッサシステム1 におけるプロセッサ状態管理装置2 2 の構成例を示す図である。

【図4】(a) および(b) は、プロセッサ状態管理装置2 2 の動作の一例を説明する図である。

【図5】(a) および(b) は、プロセッサ状態管理装

10

20

30

40

50

23

置22の動作の他の一例を説明する図である。

【図6】パケット50の構成を示す図である。

【図7】プロセッサ30～32がfork命令を解釈実行する手順を示す図である。

【図8】プロセッサ30～32がunlock命令を解釈実行する手順を示す図である。

【図9】プロセッサの状態とタスクの状態とを説明する図である。

【図10】1から4までの和を二分木に基づいて計算するプログラムの手順を示す図である。

【図11】図10に示すプログラムの処理の内容を模式的に示した図である。

【図12】従来のプロセッサ割当方法の動作を説明する図である。

【図13】タスクが中粒度～粗粒度である場合における、タスクの処理時間とオーバヘッドの処理時間とを示

24

すタイムチャートである。

【図14】タスクが細粒度である場合における、タスクの処理時間とオーバヘッドの処理時間とを示すタイムチャートである。

【符号の説明】

1 マルチプロセッサシステム

2 主記憶装置

10～12 要素プロセッサユニット

20 共有キャッシュ

10 21 ネットワーク

22 プロセッサ状態管理装置

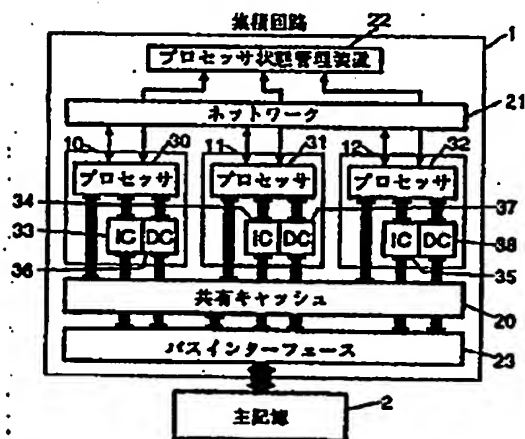
23 バスインターフェース

30～32 プロセッサ

33～35 命令キャッシュ(IC)

36～38 データキャッシュ(DC)

【図1】



【図9】

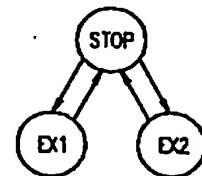
プロセッサ



IDLE アイドル

RUN 実行中

タスク

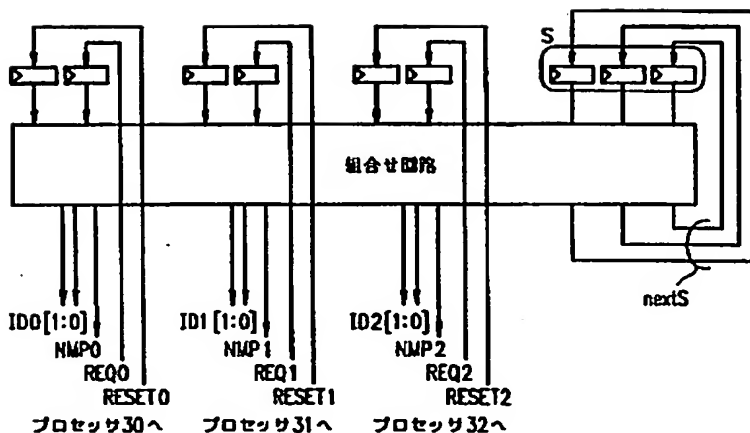


STOP 実行待ちまたは実行済

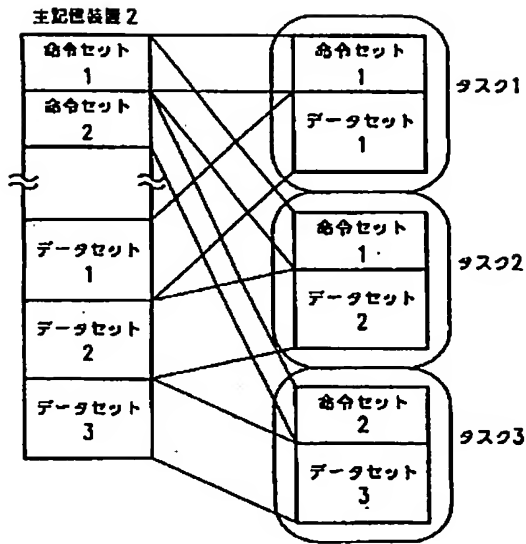
EX1 実行中(他タスク中断なし)

EX2 実行中(他タスク中断あり)

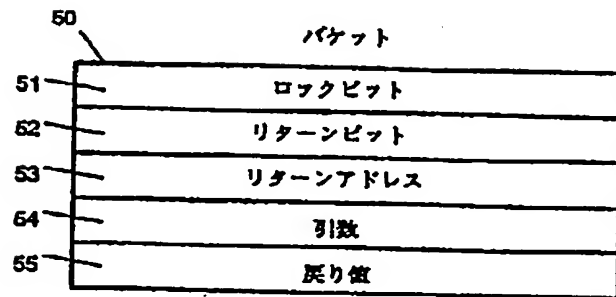
【図3】



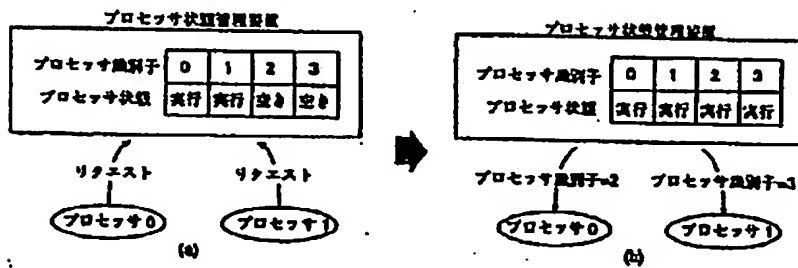
【 図2 】



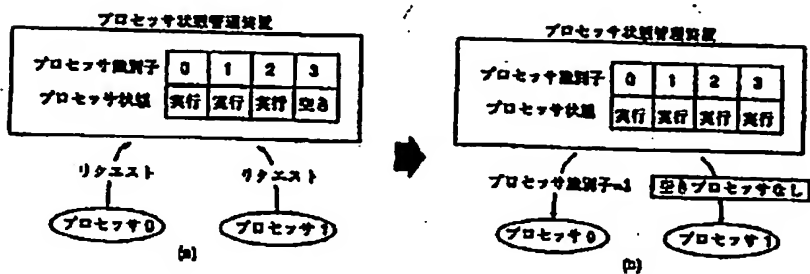
【 図6 】



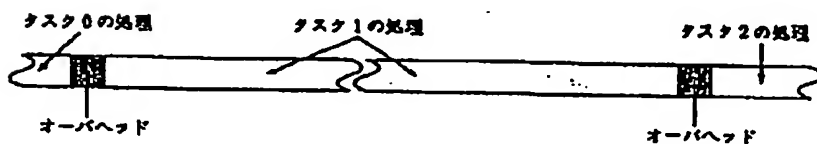
【 図4 】



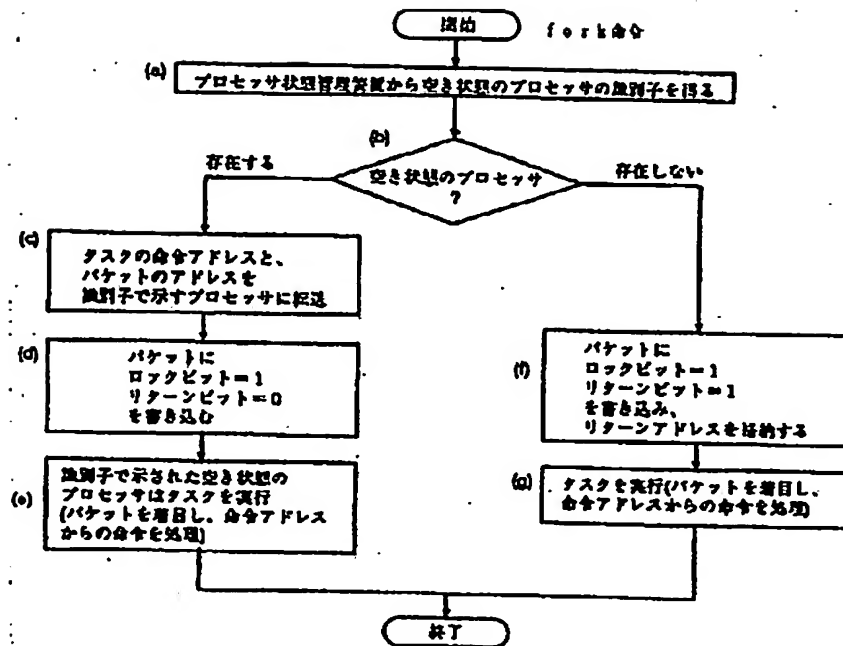
【 図5 】



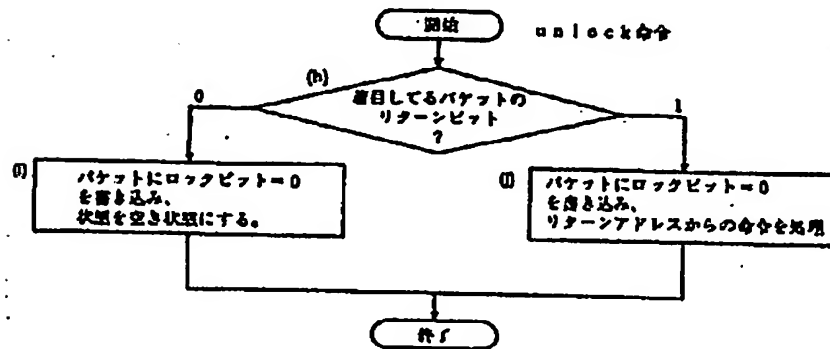
【 図13 】



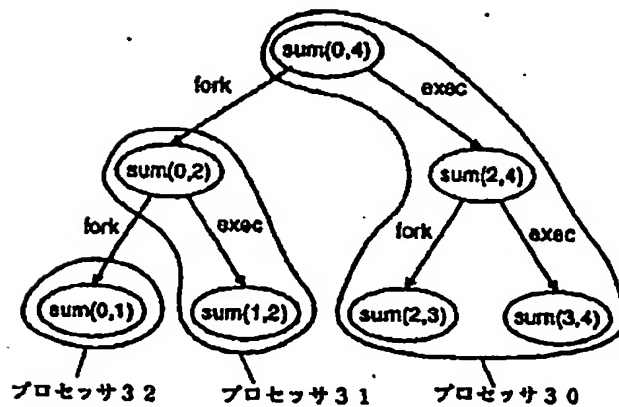
【 図7 】



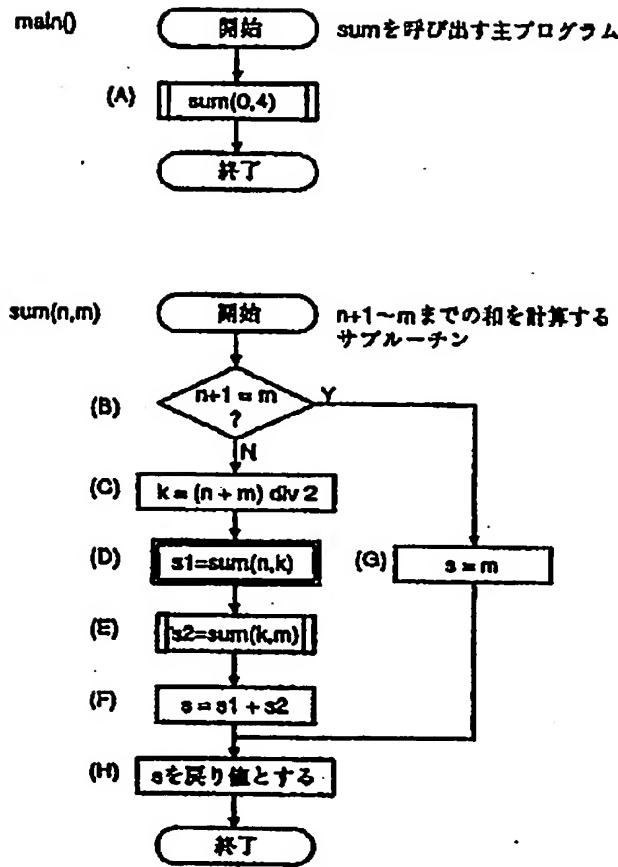
【 図8 】



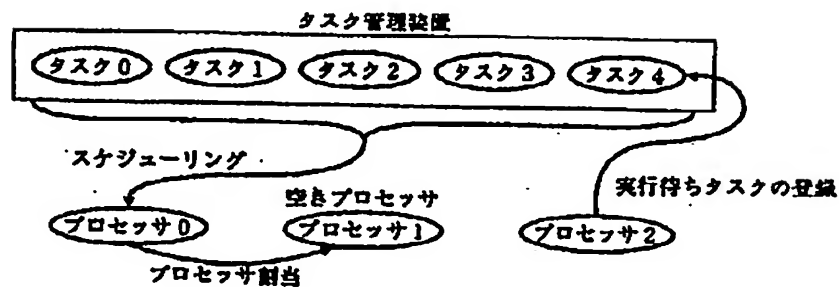
【 図11 】



【 図10 】



【 図12 】



【 図14 】

